

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARCOS AMORIM ROSSI DE CARVALHO

ÁRVORES BINÁRIAS DE BUSCA BALANCEADAS IMPLEMENTADAS EM UM
AMBIENTE MÓVEL

RIO DE JANEIRO
2020

MARCOS AMORIM ROSSI DE CARVALHO

ÁRVORES BINÁRIAS DE BUSCA BALANCEADAS IMPLEMENTADAS EM UM
AMBIENTE MÓVEL

Trabalho de conclusão de curso de graduação
apresentado ao Departamento de Ciência da
Computação da Universidade Federal do Rio
de Janeiro como parte dos requisitos para ob-
tenção do grau de Bacharel em Ciência da
Computação.

Orientador: Prof. Paulo Roma Cavalcanti D.Sc

Co-orientador:

RIO DE JANEIRO

2020

CIP - Catalogação na Publicação

C331? Carvalho, Marcos Amorim Rossi de
Árvores binárias de busca balanceadas
implementadas em ambiente móvel / Marcos Amorim
Rossi de Carvalho. -- Rio de Janeiro, 2020.
39 f.

Orientador: Paulo Roma Cavalcanti.
Trabalho de conclusão de curso (graduação) -
Universidade Federal do Rio de Janeiro, Instituto
de Matemática, Bacharel em Ciência da Computação,
2020.

1. Árvores balanceadas. 2. Ambiente móvel. 3.
Interface interativa. I. Cavalcanti, Paulo Roma,
orient. II. Título.

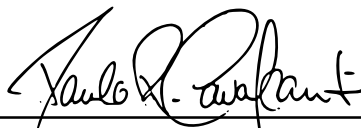
MARCOS AMORIM ROSSI DE CARVALHO

ÁRVORES BINÁRIAS DE BUSCA BALANCEADAS IMPLEMENTADAS EM UM
AMBIENTE MÓVEL

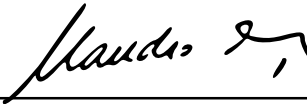
Trabalho de conclusão de curso de graduação
apresentado ao Departamento de Ciência da
Computação da Universidade Federal do Rio
de Janeiro como parte dos requisitos para ob-
tenção do grau de Bacharel em Ciência da
Computação.

Aprovado em 02 de julho de 2020

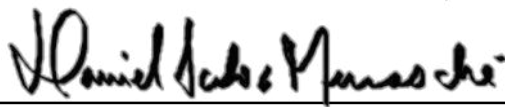
BANCA EXAMINADORA:



Prof. Paulo Roma Cavalcanti, D.Sc



Prof. Claudio Esperança, D.Sc



Prof. Daniel Sadoc Menasche, Ph.D

AGRADECIMENTOS

Gostaria de agradecer à todos os meus professores, especialmente ao professor Paulo Roma pela orientação e paciência.

Aos meus colegas de curso, que tornaram a experiência na faculdade algo mais agradável e divertido.

À minha família que me apoiou em todos os momentos dessa longa jornada.

E aos meus amigos de colégio, que mesmo cursando outras disciplinas, compartilharam suas experiências e bons momentos comigo.

RESUMO

Árvores binárias de busca balanceadas são estruturas de dados amplamente usadas para listagem ordenada e mutável de dados. O objetivo deste trabalho é apresentar uma interface intuitiva de como funciona uma árvore binária de busca balanceada e comparar diferentes tipos de árvores.

Palavras-chave: árvores binárias de busca balanceadas. ambiente móvel.

ABSTRACT

Balanced search binary trees are widely used data structures for orderly and mutable listing of data. The goal of this work is to present an intuitive interface of how a balanced binary search tree works and to compare different types of trees.

Keywords: balanced binary search trees. mobile environment.

LISTA DE ILUSTRAÇÕES

Figura 1 – Anatomia de uma árvore binária.	9
Figura 2 – Anatomia de uma árvore binária de busca.	10
Figura 3 – fatores de balanceamento de uma árvore AVL (em verde).	13
Figura 4 – Rotação simples à esquerda no nó 3.	14
Figura 5 – Rotação dupla à esquerda nos nós 3 e 5.	14
Figura 6 – Exemplo de uma árvore rubro-negra.	15
Figura 7 – Pseudocódigo de uma função de rebalanceamento de uma árvore rubro-negra após inserção.	16
Figura 8 – Rebalanceamento de uma árvore rubro-negra após uma inserção de um nó z	17
Figura 9 – Pseudocódigo de uma função de rebalanceamento de uma árvore rubro-negra após remoção.	18
Figura 10 – Pseudocódigo de uma função de rebalanceamento de uma árvore rubro-negra após remoção.	19
Figura 11 – Inserção de um nó em uma árvore AA.	21
Figura 12 – Remoção de um nó em uma árvore AA.	22
Figura 13 – Rotações zig e zag em uma árvore splay.	23
Figura 14 – Rotações zig-zig e zag-zag em uma árvore splay.	24
Figura 15 – Rotação zig-zag em uma árvore splay.	24
Figura 16 – Tela inicial do aplicativo	27
Figura 17 – Tela de Ajuda	28
Figura 18 – Botões deslocados após abertura do teclado virtual	29
Figura 19 – Exemplo de árvore AVL	30
Figura 20 – Exemplo de árvore Rubro-Negra com os mesmos nós da árvore da figura anterior.	31
Figura 21 – Exemplo de uma busca com sucesso em uma árvore AVL	31
Figura 22 – Exemplo de uma busca sem sucesso em uma árvore AVL	32
Figura 23 – Detalhes de um nó após um toque prolongado	33
Figura 24 – Exemplo de árvore AA	34
Figura 25 – Exemplo de uma árvore AVL com altura 6	35
Figura 26 – Exemplo de uma árvore AVL com ampliação dos nós	36

SUMÁRIO

1	INTRODUÇÃO	8
2	CONCEITOS BÁSICOS	9
3	ÁRVORES BALANCEADAS	13
3.1	ÁRVORE AVL	13
3.1.1	Rotações	13
3.2	ÁRVORE RUBRO-NEGRA	14
3.2.1	Rotações e recolorações	16
3.3	ÁRVORE AA	19
3.4	ÁRVORE SPLAY	22
3.4.1	Splay	23
4	IMPLEMENTAÇÃO	26
5	CONCLUSÃO	37
5.1	TRABALHOS FUTUROS	37
	REFERÊNCIAS	39

1 INTRODUÇÃO

As árvores binárias de busca (BST) são estruturas de armazenamentos de dados que permitem uma rápida pesquisa de itens na memória, como também adição e remoção de itens. Árvores podem ser usadas para implementar conjuntos dinâmicos de itens ou tabelas de consulta que permitem localizar um item por sua chave (por exemplo, encontrar o número de telefone de uma pessoa pelo nome) (WIKIPEDIA, 2019a). Árvores são estruturas de dados que combinam a flexibilidade da inserção em uma lista encadeada com a eficiência de uma busca em um vetor ordenado (SEDGEWICK; WAYNE, 2011, p. 396).

Árvores binárias são comumente usadas por bibliotecas e outras estruturas, como dicionários, mapas (como no caso da linguagem C++ (CPLUSPLUS, 2019)), filas de prioridades, bem como para escalonamento de processos que, a partir da versão 2.6.23 do Linux, foram implementados usando uma árvore rubro-negra (WIKIPEDIA, 2019b; CORBET, 2007).

Uma árvore binária de busca balanceada, também chamada de árvore binária de busca auto-balanceada, é uma denominação para qualquer árvore que mantém sua altura reduzida automaticamente para diminuir o custo de acesso, evitando assim os piores casos de uma árvore binária de busca não balanceada.

Existem diversos tipos de árvores binárias balanceadas, diferidas pelas suas características e algoritmos de balanceamento e consequentemente suas representações. Estas diferenças ocasionam maior ou menor performance na inserção e remoção de nós.

O objetivo deste trabalho é estudar algoritmos de estruturas de dados e criar uma interface interativa, com efeito didático, para a visualização das estruturas e suas diferenças e semelhanças. Foi escolhido o ambiente de dispositivos móveis com sistema operacional iOS, onde a tela maior de um iPad é recomendada para a utilização da interface.

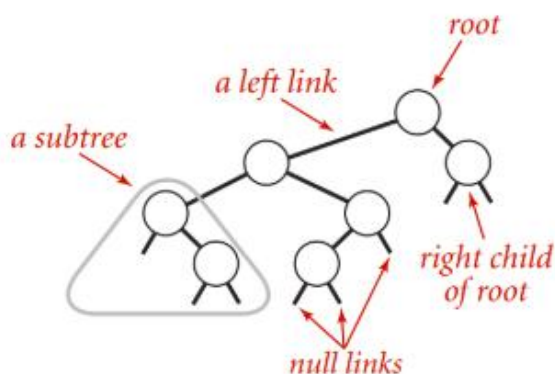
Neste trabalho serão abordados alguns tipos de árvores diferentes, demonstrando em cada uma delas as operações de busca, inserção e remoção de nós. As árvores estudadas serão: árvores AVL, Rubro-negras, AA e Splay.

2 CONCEITOS BÁSICOS

Neste capítulo serão comentados os conceitos básicos de árvores binárias de busca e as vantagens de uma árvore balanceada.

Para começar, iremos definir a terminologia básica. Árvores são estruturas de dados compostas por nós e contêm ligações que podem ser nulas, ou referências a outros nós. Em uma árvore binária temos a restrição de que todo nó só pode ser apontado por um outro nó, chamado de pai (exceto o nó chamado de raiz, que não é apontado por nenhum outro nó), e cada nó tem exatas duas ligações, podendo ser à esquerda ou à direita, que apontam para nós chamados de filhos à esquerda ou à direita respectivamente (SEDGEWICK; WAYNE, 2011, p. 396).

Figura 1 – Anatomia de uma árvore binária.



Fonte: Sedgewick e Wayne (2011, p. 396)

Podemos visualizar as ligações dos nós como um ponteiro para uma árvore binária cuja raiz é o nó referenciado. Essa árvore é chamada subárvore e possui as mesmas características da árvore binária da qual faz parte. Uma árvore também pode não conter nenhum elemento, e é chamada de árvore vazia. Um nó que tem suas duas ligações apontando para árvores vazias é chamado de folha.

uma chave desejada. A altura de uma árvore é dada pela quantidade de nós no maior caminho da raiz até uma folha. Se uma árvore tem n nós, a sua altura mínima é $\log(n)$ arredondada para baixo. A cada utilização da árvore, o custo de acesso deve ser mantido na mesma ordem de grandeza de uma árvore ótima, ou seja, $O(\log n)$ (SZWARCFITER; MARKENZON, 1994, p. 127).

Para manter a altura de uma árvore semelhante à de uma árvore mínima, são aplicados algoritmos de balanceamento após cada modificação na árvore (inserção ou remoção de um nó). Algoritmos de balanceamento podem realizar transformações na árvore chamadas de rotações. Rotações são reestruturações de uma árvore que movimentam nós para reduzir e aumentar a altura de subárvores que se encontram desbalanceadas.

Uma árvore completa é uma árvore com um número n de nós e altura h que segue a seguinte fórmula (SZWARCFITER; MARKENZON, 1994, p. 94).

$$2^h \leq n \leq 2^{h+1} - 1 \quad (2.1)$$

Árvores completas são aquelas que minimizam o número de comparações efetuadas no pior caso para uma busca. Balanceamento é um algoritmo que tem como objetivo manter a altura da árvore na mesma ordem de grandeza que uma árvore completa com o mesmo número de nós, isto é $O(\log n)$ (SZWARCFITER; MARKENZON, 1994, p. 128).

Todas as árvores binárias de busca usam métodos muito parecidos para buscar, inserir, e remover nós, mudando apenas suas operações de balanceamento de acordo com o tipo de árvore usado.

A operação de busca funciona percorrendo os nós da árvore começando pela raiz. Essa operação é realizada de acordo com seu identificador que compara com o de seus filhos e decide se a busca encerra ou continua. Caso o identificador do nó que está sendo procurado seja igual ao do nó corrente da árvore, a busca é encerrada pois foi encontrado o nó procurado. Caso o identificador do nó que está sendo procurado seja menor que o identificador do nó da árvore, a busca continua no filho esquerdo (caso o nó atual tenha um filho à esquerda). Caso o nó não tenha um filho à esquerda, a busca é encerrada com o nó procurado não pertencendo a árvore. Para o caso de o identificador do nó procurado ser maior que o do nó da árvore, a busca continua no filho direito ou encerra caso este não exista.

Para a operação de inserção, é feita uma operação de busca pelo nó que será inserido. Caso o nó buscado seja encontrado na árvore, a operação de inserção é encerrada. Caso contrário, o novo nó é adicionado como filho do último nó encontrado na busca. Caso o identificador do nó a ser inserido seja menor que o último nó da busca, ele é inserido como filho esquerdo. Caso contrário, ele é inserido como filho direito.

Para a operação de remoção também é feita uma busca para achar o nó a ser removido. Caso o nó não pertença à árvore a operação é finalizada. Caso contrário, o nó é removido e é colocado seu sucessor ou antecessor no seu lugar. Para achar o sucessor de um nó, basta apenas seguir os filhos esquerdos do seu filho direito. De maneira similar ao sucessor, para encontrar o antecessor, é necessário seguir os filhos direitos do filho esquerdo do nó.

Após uma inserção ou remoção, é feita a checagem de desbalanceamento da árvore. Como essas funções usadas na árvore são recursivas, ou seja, uma função que chama a si própria, a função de balanceamento é feita em todos os nós entre o nó afetado até o nó raiz. Para garantir o balanceamento da árvore, são efetuadas rotações que servem para reduzir a altura a árvore. Cada tipo de árvore possui diferentes condições de balanceamento, cada uma com suas vantagens e desvantagens, que serão mais detalhadas nas próximas seções.

3 ÁRVORES BALANCEADAS

Neste capítulo serão discutidos os diferentes tipos de árvores binárias de busca balanceadas, suas condições de balanceamento, suas similaridades e suas diferenças.

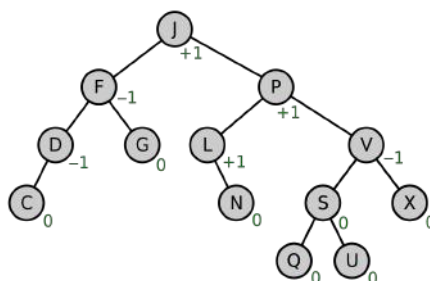
3.1 ÁRVORE AVL

Árvores AVL receberam esse nome devido a terem sido introduzidas por Adel'son-Vel'skii e Landis em 1962. Uma árvore AVL tem como propriedade, para balanceamento, o fato de que a diferença de altura entre os filhos de cada nó interno não pode ser maior que 1, garantindo assim que a altura de uma árvore AVL com n nós seja de $O(\log n)$ (GOODRICH; TAMASSIA, 2001, p. 152-153).

Cada nó possui uma propriedade extra chamada de fator de balanceamento, que é dado pela altura da sua subárvore esquerda, menos a altura da sua subárvore direita. Nós balanceados são nós onde o fator de balanceamento vale -1, 0 ou 1.

Após uma inserção ou remoção, na checagem de desbalanceamento, se o fator de balanceamento de um nó tem módulo maior que 1, este nó está desbalanceado e a árvore precisa realizar uma rotação em seus nós para ser balanceada.

Figura 3 – fatores de balanceamento de uma árvore AVL (em verde).



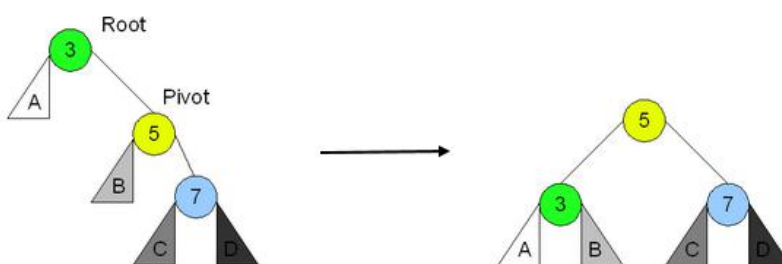
Fonte: Wikipedia (AVL Tree)

3.1.1 Rotações

Conforme observado anteriormente, para garantir que uma árvore se mantenha balanceada, é necessário realizar uma rotação sempre que um nó desbalanceado é encontrado. Em uma árvore AVL, são necessárias quatro rotações para manter o balanceamento. Essas rotações são chamadas de rotações simples à esquerda, simples à direita, dupla à esquerda e dupla à direita, sendo uma rotação à esquerda simétrica a uma rotação à direita.

Uma rotação simples à esquerda deve ser feita quando o fator de balanceamento de um nó é igual a 2 e o fator de balanceamento de seu filho direito é igual a 1. O nó filho direito deve tomar o lugar do pai e o nó desbalanceado deve se tornar o filho esquerdo do seu antigo filho direito. Simetricamente, é realizada uma rotação à direita caso o fator de balanceamento seja -2 e seu filho esquerdo tenha fator de balanceamento -1.

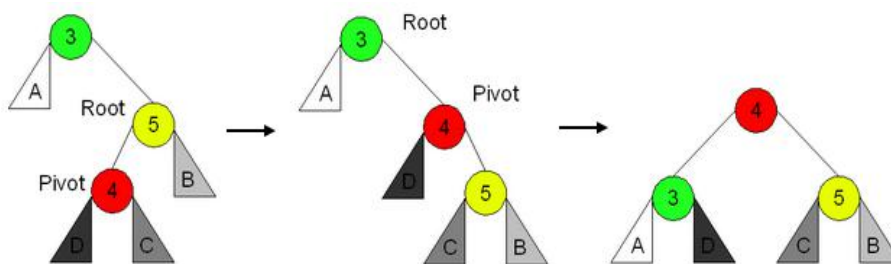
Figura 4 – Rotação simples à esquerda no nó 3.



Fonte: Wikipedia (AVL Tree)

Uma rotação dupla à esquerda deve ser efetuada quando o fator de balanceamento de um nó é igual a 2 e o fator de balanceamento do filho direito é igual a -1. Nesse caso devemos aplicar uma rotação à direita no nó filho direito e, em seguida, uma rotação à esquerda no nó desbalanceado. Simetricamente, é realizada uma rotação à direita caso o fator de balanceamento seja -2 e seu filho esquerdo tenha fator de balanceamento 1.

Figura 5 – Rotação dupla à esquerda nos nós 3 e 5.



Fonte: Wikipedia (AVL Tree)

3.2 ÁRVORE RUBRO-NEGRA

Uma árvore rubro-negra, também conhecida como árvore vermelho-preto, teve sua estrutura original criada em 1972, por Rudolf Bayer, sendo ela um caso especial de árvore B, que é uma árvore onde cada nó pode ter mais de uma chave e consequentemente mais de dois filhos. Foi chamada inicialmente de "Árvores Binárias B Simétricas", mas adquiriu seu nome moderno em um artigo de 1978 escrito por Leonidas J. Guibas e Robert Sedgwick (CORMEN et al., 2002, p. 241).

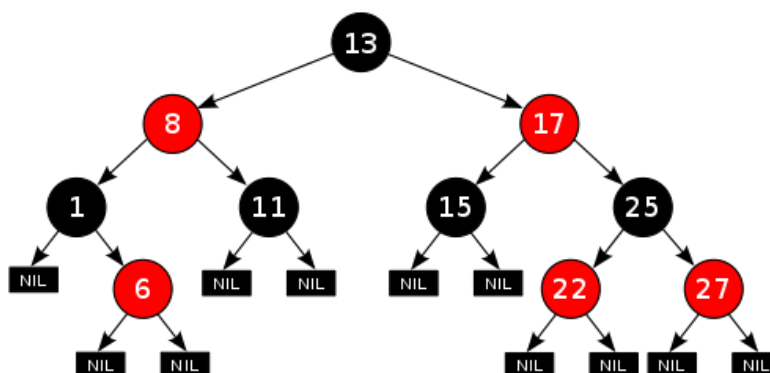
Uma árvore rubro-negra é uma árvore de busca com um atributo extra em cada nó, utilizado para guardar sua cor, que pode ser vermelho ou preto. Restringindo como os nós podem ser coloridos em qualquer caminho desde a raiz até uma folha, as árvores rubro-negras asseguram que nenhum desses caminhos será maior que duas vezes o comprimento de qualquer outro, de forma que a árvore é aproximadamente balanceada (CORMEN et al., 2002, p. 220).

Uma característica adicional de uma árvore rubro-negra é a de que se algum nó não tiver filho esquerdo ou direito, o campo do ponteiro correspondente do nó conterá o valor NIL, e esses valores serão tratados como folhas da árvore.

O balanceamento de uma árvore rubro-negra é dependente da cor de seus nós e a árvore deve ter as seguintes propriedades para ser considerada rubro-negra:

- a) Todo nó é vermelho ou preto.
- b) A raiz é preta.
- c) Todas as folhas (NIL) são pretas.
- d) Ambos os filhos de todos os nós vermelhos são pretos.
- e) Todo caminho de um dado nó para qualquer de seus nós folha descendentes contém o mesmo número de nós pretos.

Figura 6 – Exemplo de uma árvore rubro-negra.



Fonte: Wikipedia (Red-black_tree)

A inserção de elementos em uma árvore rubro-negra é semelhante a uma inserção em uma árvore binária, porém, substituindo uma folha nula e colorindo o novo nó de vermelho mantendo suas folhas pretas. Similar à uma árvore AVL, o algoritmo para manter as propriedades da árvore rubro-negra é executado da folha até a raiz, logo após uma inserção ou remoção.

3.2.1 Rotações e recolorações

A seguir iremos demonstrar como uma árvore rubro-negra pode ficar desbalanceada e como realizar rotações e recolorações para rebalancear a árvore. No pseudocódigo a seguir são mostrados os casos em que é necessário realizar um rebalanceamento depois de uma inserção, sendo z o nó atual na checagem e p o pai de um nó.

Figura 7 – Pseudocódigo de uma função de rebalanceamento de uma árvore rubro-negra após inserção.

```

RB-INSERT-FIXUP( $T, z$ )
1  while  $z.p.color == RED$ 
2      if  $z.p == z.p.p.left$ 
3           $y = z.p.p.right$ 
4          if  $y.color == RED$ 
5               $z.p.color = BLACK$                                 // case 1
6               $y.color = BLACK$                                 // case 1
7               $z.p.p.color = RED$                                 // case 1
8               $z = z.p.p$                                         // case 1
9          else if  $z == z.p.right$ 
10              $z = z.p$                                           // case 2
11             LEFT-ROTATE( $T, z$ )                               // case 2
12              $z.p.color = BLACK$                                 // case 3
13              $z.p.p.color = RED$                                 // case 3
14             RIGHT-ROTATE( $T, z.p.p$ )                          // case 3
15         else (same as then clause
                with “right” and “left” exchanged)
16      $T.root.color = BLACK$ 

```

Fonte: Cormen (2002, p. 226)

Para entendermos como funciona a função de rebalanceamento veremos quais os casos em que uma árvore pode violar as propriedades rubro-negra e como proceder para recuperar essas propriedades. Como um nó ao ser inserido é colorido de vermelho, as únicas propriedades que ele pode violar são a de que a raiz sempre é preta ou de que um nó vermelho só pode ter filhos pretos. Caso o nó inserido seja a raiz, basta apenas recolorir ele de preto. Caso ele seja filho de outro nó vermelho, veremos os casos separados de como recolorir e rotacionar os nós. A figura a seguir mostra os casos citados no pseudocódigo anterior e mostra como a árvore se reajusta.

Figura 9 – Pseudocódigo de uma função de rebalanceamento de uma árvore rubro-negra após remoção.

```

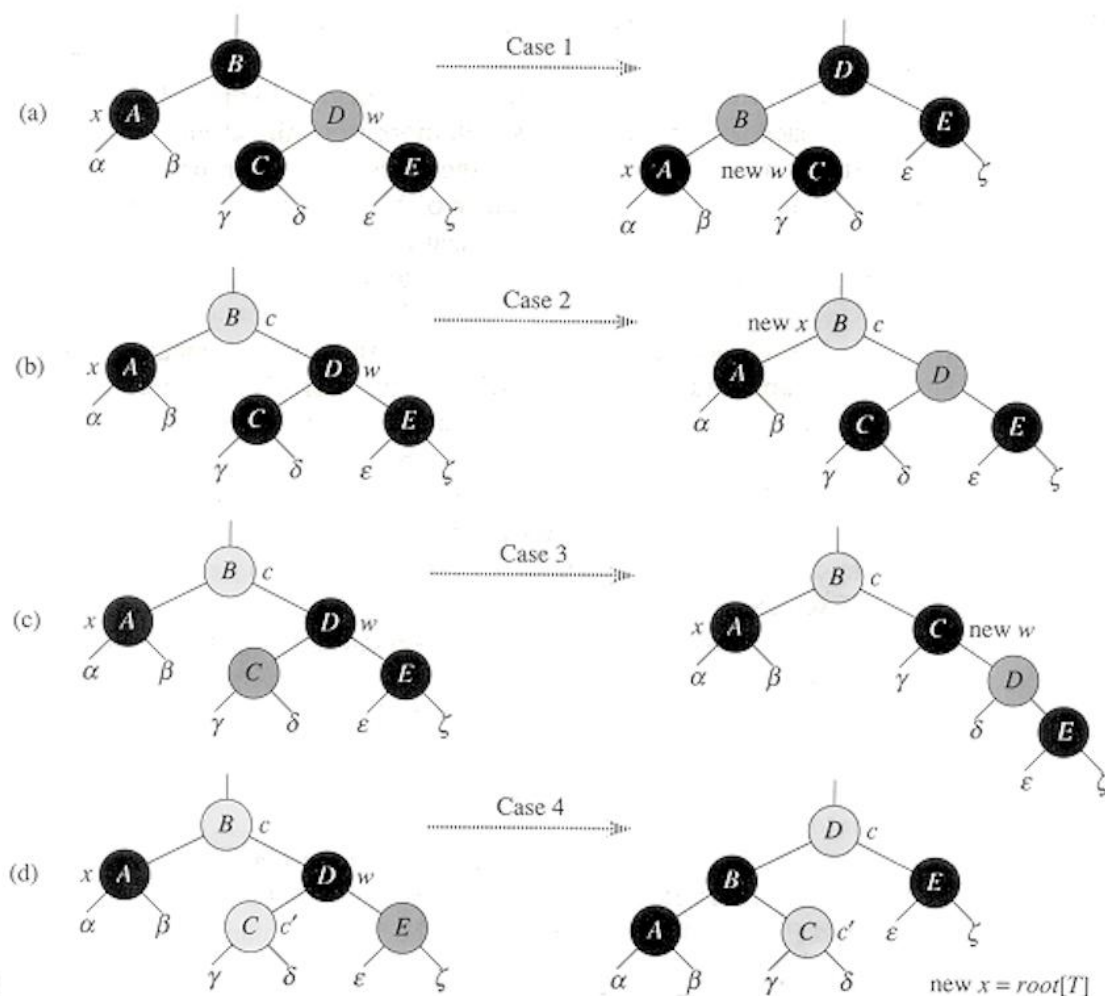
RB-DELETE-FIXUP(T, x)
1  while x ≠ T.root and x.color == BLACK
2      if x == x.p.left
3          w = x.p.right
4          if w.color == RED
5              w.color = BLACK                // case 1
6              x.p.color = RED                // case 1
7              LEFT-ROTATE(T, x.p)          // case 1
8              w = x.p.right                // case 1
9          if w.left.color == BLACK and w.right.color == BLACK
10             w.color = RED                // case 2
11             x = x.p                      // case 2
12         else if w.right.color == BLACK
13             w.left.color = BLACK          // case 3
14             w.color = RED                // case 3
15             RIGHT-ROTATE(T, w)           // case 3
16             w = x.p.right                // case 3
17             w.color = x.p.color          // case 4
18             x.p.color = BLACK            // case 4
19             w.right.color = BLACK        // case 4
20             LEFT-ROTATE(T, x.p)          // case 4
21             x = T.root                  // case 4
22         else (same as then clause with “right” and “left” exchanged)
23     x.color = BLACK

```

Fonte: Cormem (2002, p. 235)

Assim como na inserção, podemos identificar no algoritmo os casos em que são necessárias rotações e recolorações, conforme ilustrado na figura 10 abaixo. Nesta figura, podemos ver os nós antes e depois das transformações da árvore e a cor de cada nó, sendo que os nós escurecidos têm o atributo de cor preto, nós fortemente sombreados têm o atributo de cor vermelho e nós levemente sombreados podem ter o atributo de cor preto ou vermelho.

Figura 10 – Pseudocódigo de uma função de rebalanceamento de uma árvore rubro-negra após remoção.



Fonte: Cormem (2002, p. 235)

3.3 ÁRVORE AA

Uma árvore AA é uma variação de árvores rubro-negra mais simples de codificar. Ela recebeu esse nome devido ao seu criador, Arnes Andersson. Árvores AA são semelhantes à uma árvore rubro-negra, exceto pelo fato de que os filhos à esquerda nunca podem ser vermelhos (CORMEN et al., 2002, p. 241).

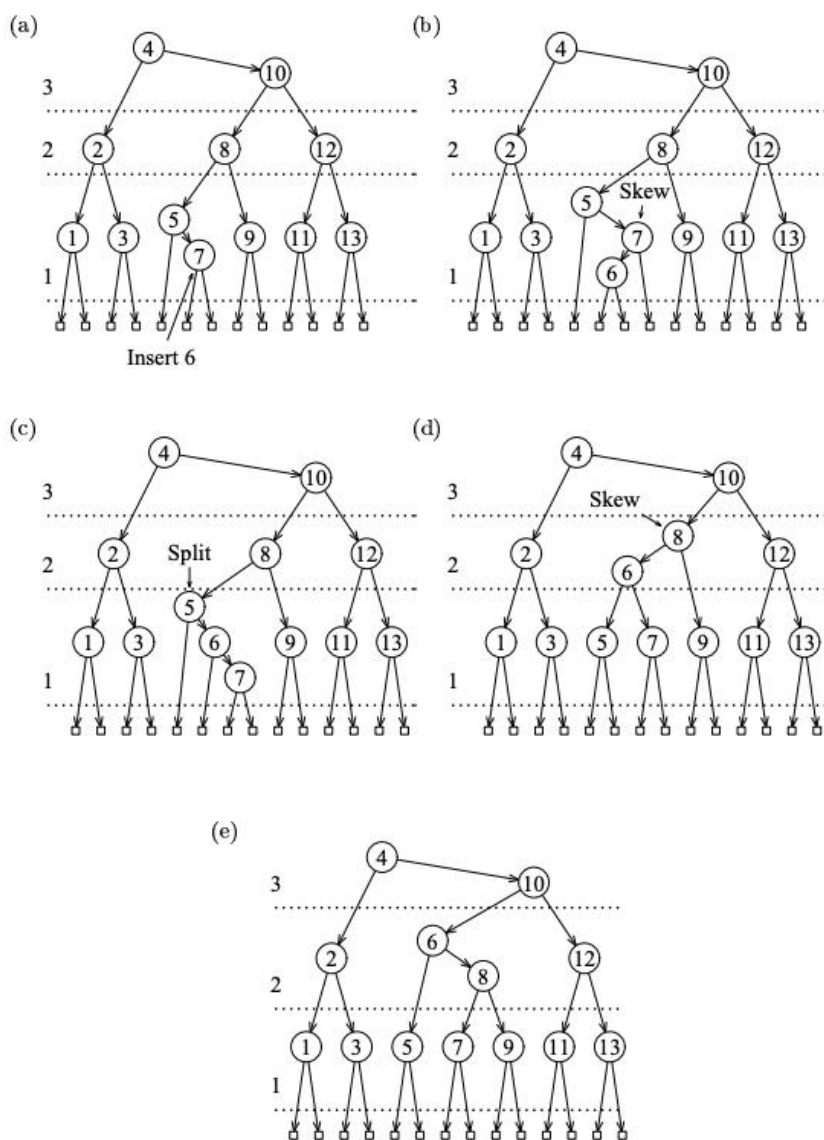
Diferentemente da árvore rubro-negra, em vez de cor, cada nó guarda informação de seu nível. As rotações de balanceamento são feitas de acordo com o nível dos nós e do nível do seu respectivo pai. Caso o nó seja vermelho, ele tem o mesmo nível de seu pai, caso o nó seja preto, ele tem um nível a menos que seu pai e se o nó for uma folha, seu nível é 1.

Como apenas os nós à direita podem ser vermelhos, há menos casos possíveis de configurações onde uma árvore pode violar a propriedade de uma árvore AA.

As rotações das árvores AA são chamadas de skew e split. Skew é uma rotação à direita que é usada quando um nó tem um filho vermelho à esquerda. Essa rotação elimina a possibilidade de haver um caminho onde um nó tenha o mesmo nível do seu filho esquerdo. Split é uma rotação à esquerda e eleva o nível do nó filho direito em 1 e é usada quando há dois nós vermelhos consecutivos à direita.

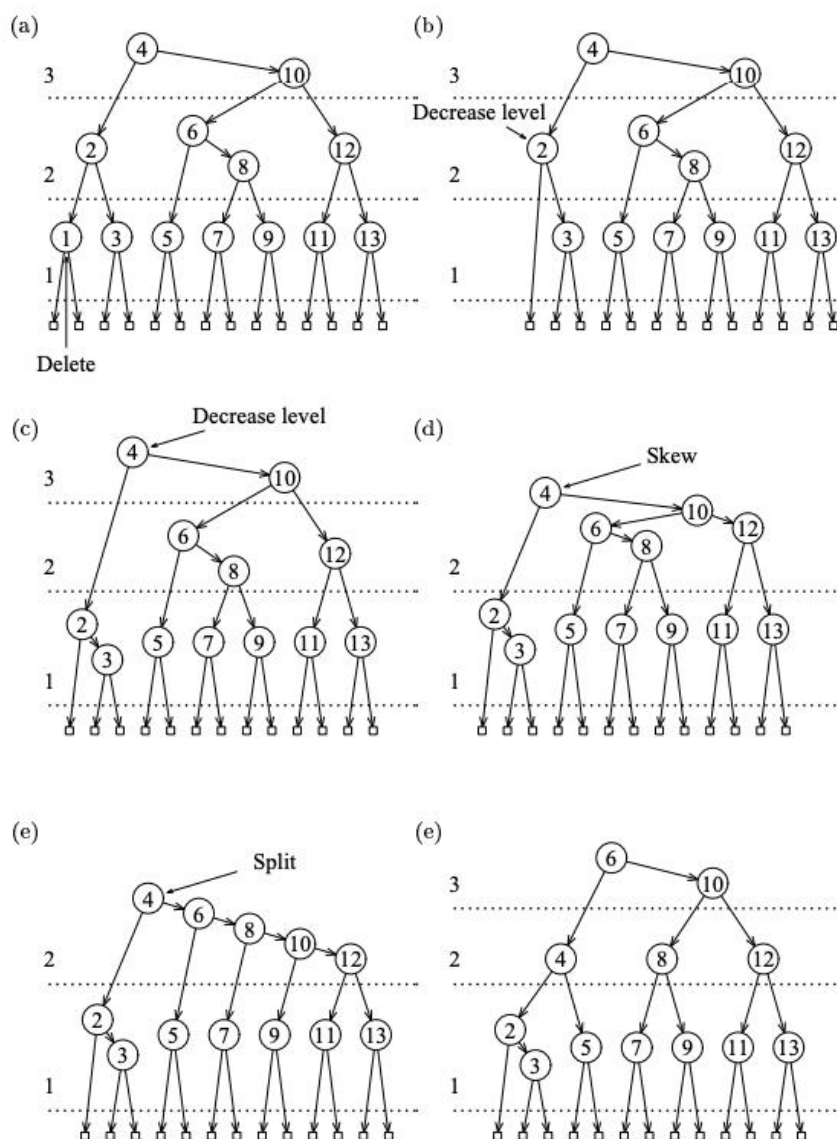
Para a inserção em uma árvore AA é necessário inserir o nó normalmente na árvore e depois para cada nó no caminho da folha até a raiz se realiza um skew seguido de split. Para a remoção, se remove normalmente, substituindo o nó excluído por uma folha. Para manter válidas as propriedades da árvore, é necessário checar cada nó, entre o excluído até a raiz, se há algum nó com um filho dois níveis abaixo do seu próprio. Se houver casos onde um nó tem um filho dois níveis abaixo, é necessário diminuir o nível do nó e realizar skew e split no nível inteiro. A seguir serão ilustradas inserções e remoções em uma árvore AA (ANDERSSON, 1993).

Figura 11 – Inserção de um nó em uma árvore AA.



Fonte: Andersson (1993)

Figura 12 – Remoção de um nó em uma árvore AA.



Fonte: Andersson (1993)

3.4 ÁRVORE SPLAY

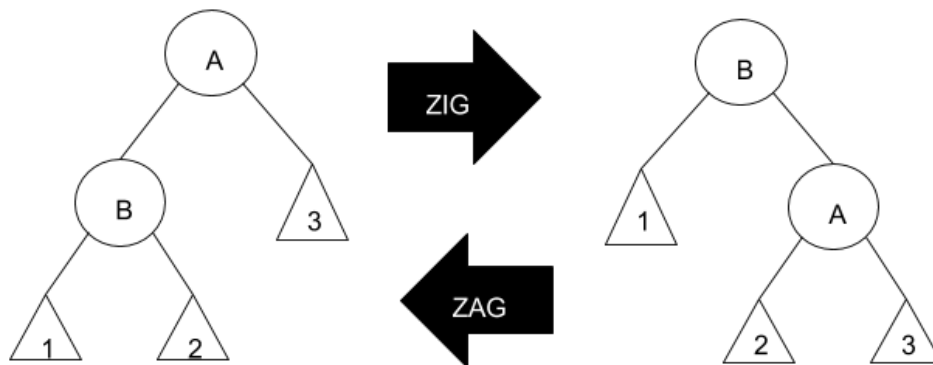
A árvore Splay é diferente das árvores anteriores por ela não ter nenhuma regra de balanceamento explícita, mas usando o conceito de mover nós recentemente acessados para raiz. Essa operação de mover um nó para a raiz é chamado de splay. Diferente das árvores anteriores, a operação splay é sempre utilizada após uma operação, ou seja, mesmo após uma simples busca, a árvore pode receber rotações em seus nós (GOODRICH; TAMASSIA, 2001, p. 185).

3.4.1 Splay

Para a operação de splay, é necessário mover um nó x para a raiz da árvore através de uma sequência específica de rotações que dependem das posições de x , seu pai e seu avô, caso este último exista. As rotações de uma árvore splay receberam o nome de zig, zag, zig-zig, zag-zag e zig-zag, onde uma rotação zig é simétrica à uma rotação zag.

Rotações zig são rotações simples onde o nó sofrendo splay é filho à esquerda de seu pai e não tem avô, e é equivalente a uma rotação simples à direita. Simetricamente uma rotação zag é equivalente a uma rotação à esquerda e é utilizada quando o nó é filho direito da raiz.

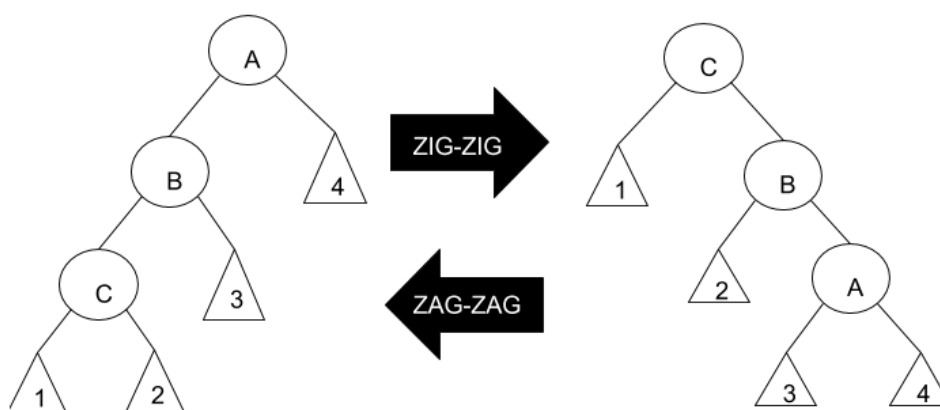
Figura 13 – Rotações zig e zag em uma árvore splay.



Fonte: Wikipedia (Árvore splay)

Uma rotação zig-zig é uma rotação onde um nó e seu pai são filhos à esquerda e são realizadas duas rotações zig. Simetricamente, uma rotação zag-zag é uma rotação que acontece quando um nó e seu pai são filhos à direita e é composta de duas rotações zag.

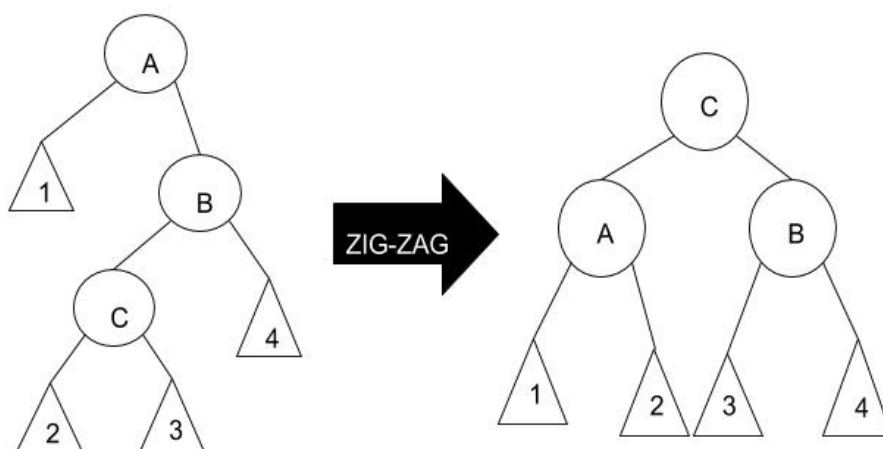
Figura 14 – Rotações zig-zig e zag-zag em uma árvore splay.



Fonte: Wikipedia (Árvore splay)

Uma rotação zig-zag é usada quando um nó e seu pai são um filho à direita e o outro um filho à esquerda e consiste em uma rotação zig seguida de uma rotação zag ou zag seguida de zig.

Figura 15 – Rotação zig-zag em uma árvore splay.



Fonte: Wikipedia (Árvore splay)

Rotações zig e zag são utilizadas quando um nó x não tem avô. As demais rotações são utilizadas quando x tem avô. Splay é a repetição dessas rotações no nó x até ele virar a raiz, ou seja, não ser filho de ninguém.

Para a remoção de um nó, primeiro aplicamos splay no nó que será removido e, da mesma maneira que em uma árvore AVL, realizamos a remoção do nó, que está na raiz da árvore, e o substituímos pelo seu antecessor ou sucessor.

Como a operação splay é a única operação usada na árvore e consiste apenas de rotações simples, a árvore pode ter alturas maiores que outras árvores balanceadas. Por exemplo, se uma sequência de nós ordenados for inserida na árvore, todos os nós seriam filhos direitos e a operação de splay faria uma rotação zag para deixar o nó na raiz, fazendo com que todos os nós inseridos ficassem à esquerda da árvore.

4 IMPLEMENTAÇÃO

Neste capítulo serão comentadas as implementações das árvores binárias de busca no ambiente iOS.

O aplicativo foi desenvolvido na linguagem Objective-C++ e escrito no XCode, ambiente de desenvolvimento de software para programas em plataformas iOS e macOS. O aplicativo é composto de quatro telas principais, representando as quatro árvores discutidas, e foram usadas bibliotecas em C++ para a estrutura de árvores e para suas rotações.

Para a estrutura usada pelo o aplicativo foi escolhido o modelo de TabBarController, onde se tem uma barra horizontal na parte inferior da tela com abas que representam suas árvores correspondentes.

Ao abrir o aplicativo, a primeira tela a ser carregada é a da árvore AVL e nela são feitas as configurações iniciais do TabBarController. A árvore AVL foi usada como tela inicial por ser a árvore mais simples entre as escolhidas.

Cada tela é dividida em duas seções, a de controle e a de visualização, usando uma estrutura SplitViewController. Na seção de controle ficam o campo de texto e os botões para modificar a árvore. Na seção de visualização é mostrada a árvore atual e onde podemos ver detalhes da árvore ou remover nós. A seção de controle pode ser escondida para melhor visualização da árvore.

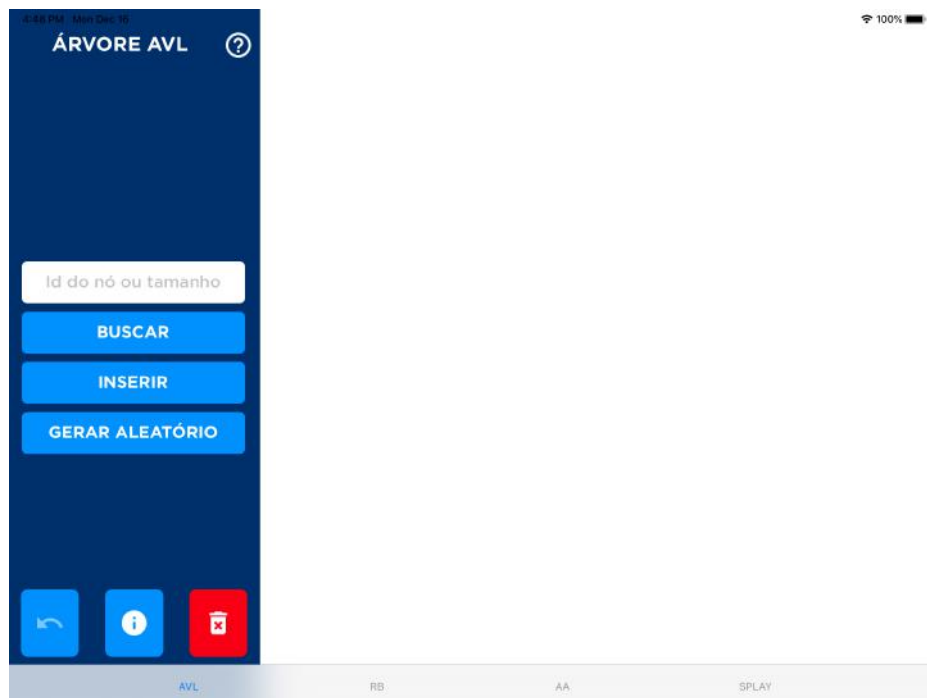
Em aparelhos menores, que não são compatíveis com o uso de um SplitViewController, a estrutura é substituída por um UINavigationController, onde a tela de controle é separada da tela de visualização. Nesses casos, a tela de visualização entra por cima da tela de controle, e recebe um botão para voltar à tela anterior.

Nas telas foram inseridas funções chamadas UITapGestureRecognizer, que são funções nativas para reconhecer um toque na tela, e ganharam diferentes funcionalidades dependendo de onde o usuário toca na tela. Se o usuário tocar em uma área de tela vazia, é ativada uma função para esconder o teclado virtual. Se o toque for em um nó, é ativada a função para a remoção do nó escolhido. Por último, são acrescentadas funções UISwipeGestureRecognizer, para esconder e mostrar a área de controle, que são ativadas quando o usuário desliza o dedo para à esquerda sob a seção de controle para escondê-la, ou deslizando o dedo para a direita para mostrar novamente a seção de controle.

A seção de controle possui um campo de texto que é usado por três botões, onde é possível: buscar por um nó, inserir um nó, ou gerar uma árvore nova com uma quantidade personalizada de nós com identificadores aleatórios. Além desses botões há mais três botões independentes do campo de texto, onde é possível desfazer a última operação feita

na árvore, obter informações da árvore ou remover todos os nós da árvore atual. A figura 16 mostra a tela do aplicativo com as seções de controle e visualização.

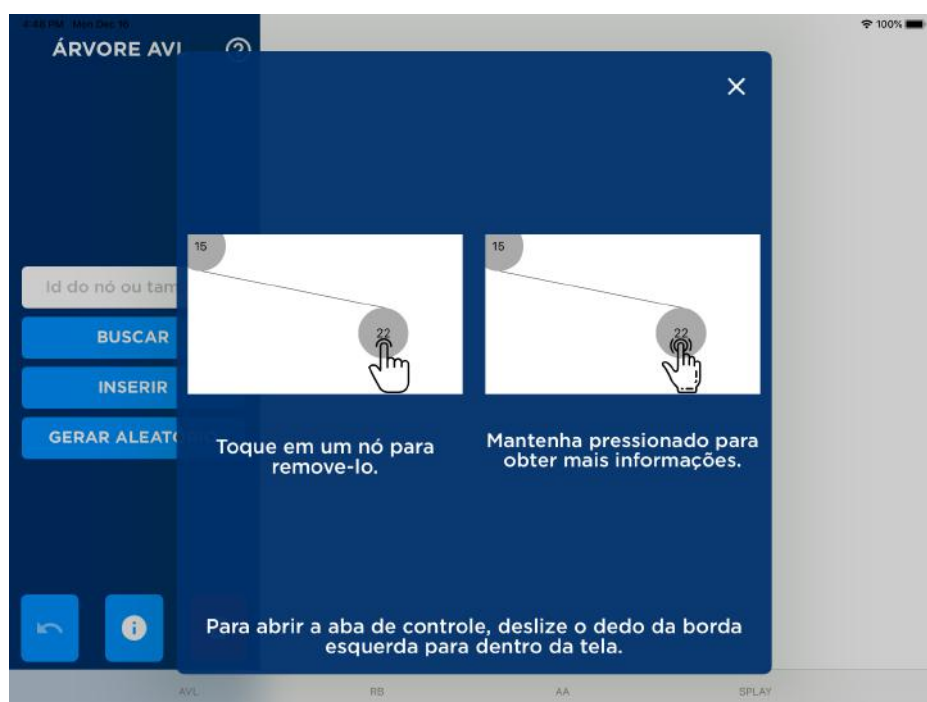
Figura 16 – Tela inicial do aplicativo



A seção de visualização possui apenas um scrollview, que é a área onde são desenhadas as árvores, e caso a árvore seja maior que a área designada, é possível ampliar e deslocar a parte da árvore que se quer melhor observar.

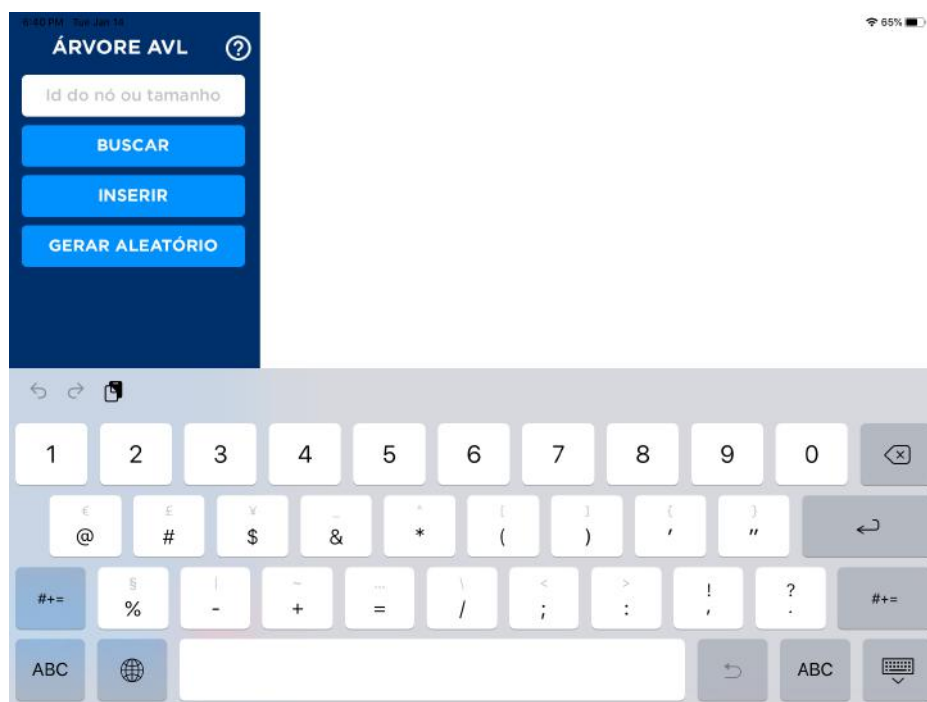
Para auxiliar na utilização do aplicativo foi adicionado um botão de ajuda no canto superior direito da seção de controle. Ao pressionar este botão é exibida uma tela mostrando os gestos usados na navegação do aplicativo e na interação do usuário com a árvore.

Figura 17 – Tela de Ajuda



Ao tocar no campo de texto o teclado virtual é aberto, possibilitando o usuário de escrever um número que será usado pelos botões. Para a busca e inserção de um nó esse número indica o identificador do nó e para a geração de uma árvore aleatória esse número indica o tamanho da árvore (quantidade de nós da árvore). Os botões e o campo de texto são deslocados para cima quando o teclado virtual é aberto a fim de não ficarem ocultos, como mostra a figura 18. Os botões e o campo de texto voltam às suas posições originais quando o teclado virtual é escondido.

Figura 18 – Botões deslocados após abertura do teclado virtual



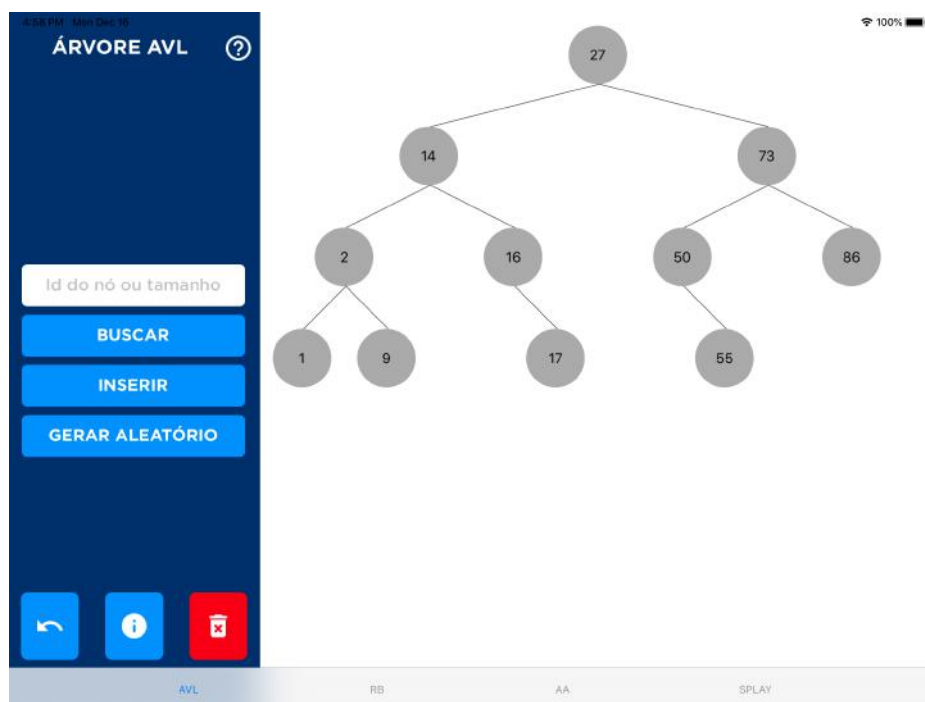
Para desenhar a árvore, primeiro é realizado um cálculo do tamanho da árvore, que é feito de acordo com a sua altura, determinando quantos nós podem existir no total e usando esses dados para estabelecer a altura e largura do scrollview onde a árvore será desenhada e o tamanho de cada nó. Após esse cálculo é chamada uma função recursiva onde são desenhados os nós individualmente.

A função de desenhar os nós recebe o nó que será desenhado na tela e as coordenadas onde ele será desenhado. Essa função sempre é chamada primeiramente na raiz da árvore e percorre os filhos dando prioridade ao filho esquerdo e em sequência desenhando seu filho direito no retorno da recursão. Para desenhar cada nó na tela é usada uma estrutura de interface replicável xib.

Cada xib é formado de uma UIView, que é uma área de tela que pode conter outros elementos, que possui uma outra UIView para representar o nó. Essa UIView interna possui um UILabel, que é uma área de texto utilizada para demonstrar o identificador de cada nó. A UIView interna é arredondada e colorida de acordo com a árvore utilizada.

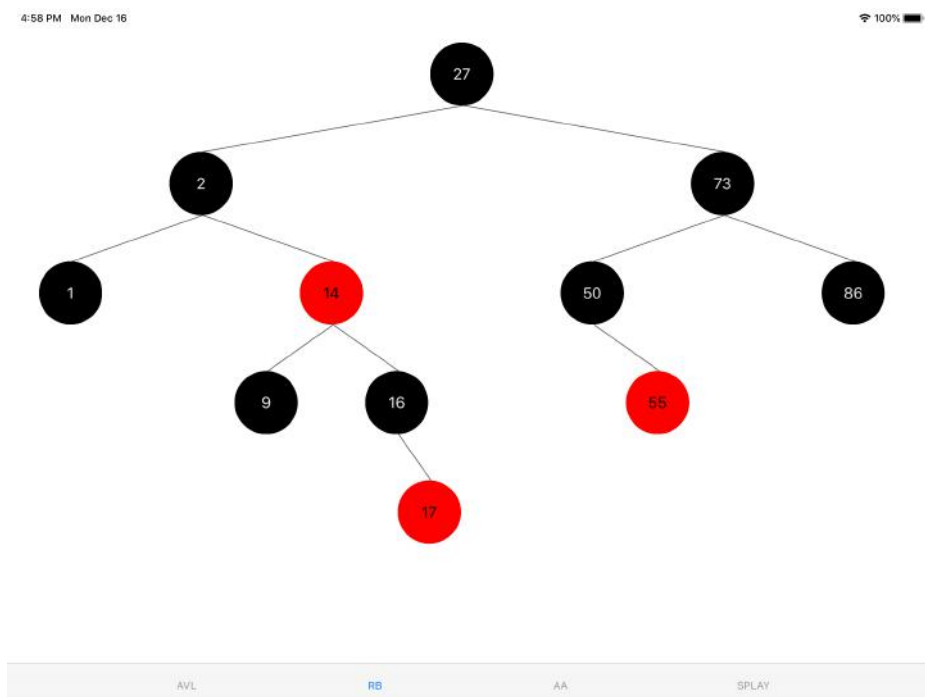
No caso das árvores rubro-negra e AA os nós são coloridos de acordo com suas informações de balanceamento, podendo ser pretos ou vermelhos. Nas árvores AVL e splay é usada a cor cinza para colorir todos os nós.

Figura 19 – Exemplo de árvore AVL



Para a comparação de árvores, seja a árvore gerada aleatoriamente ou pela inserção individual de nós, é gravada a ordem que os nós foram inseridos na árvore até uma operação de desfazer ou de remoção ser realizada. Quando o usuário trocar o tipo de árvore visualizada, se for encontrada uma sequência de nós guardada na memória e a árvore atual para o tipo escolhido estiver vazia, é criada uma nova árvore com a sequência de nós armazenada. Essa árvore facilita a comparação das diferenças e semelhanças entre as árvores escolhidas, como podemos ver na figura a seguir.

Figura 20 – Exemplo de árvore Rubro-Negra com os mesmos nós da árvore da figura anterior.



Para a busca de um nó é usada a cor verde para simbolizar o caminho percorrido. Caso o nó seja encontrado ele é circulado de verde. Caso o nó não seja encontrado é exibido um alerta dizendo que o nó não se encontra na árvore.

Figura 21 – Exemplo de uma busca com sucesso em uma árvore AVL

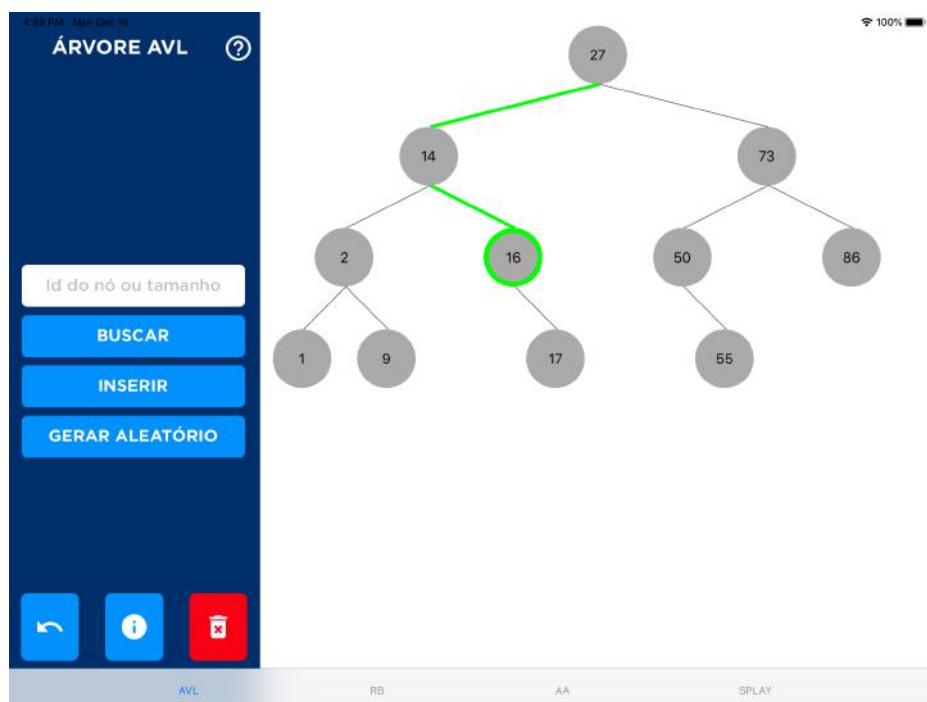


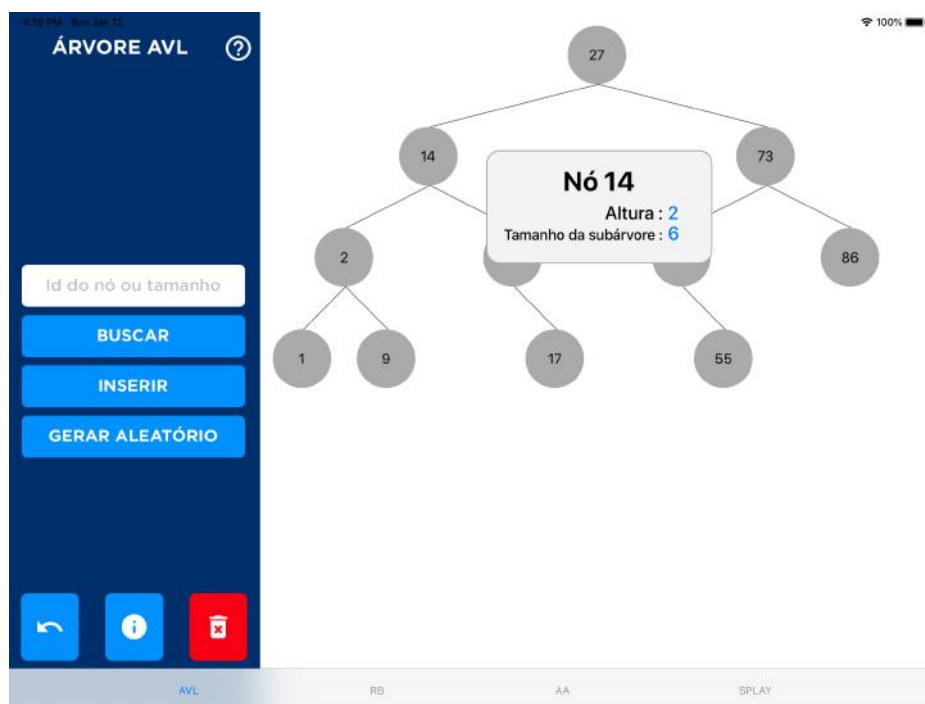
Figura 22 – Exemplo de uma busca sem sucesso em uma árvore AVL



A cada iteração da função de desenhar um nó, este tem sua cor determinada (vermelho, preto ou cinza), tem seu identificador escrito e tem seus cantos arredondados, transformando a área em um círculo. É também nesse momento que o UILabel tem acrescentado um UITapGestureRecognizer, que é usado para chamar a função de remoção do nó. Também é criado um UILongPressGestureRecognizer, que é uma função que detecta um toque prolongado na tela, usado para exibir as informações do nó na tela, como podemos ver na figura 23.

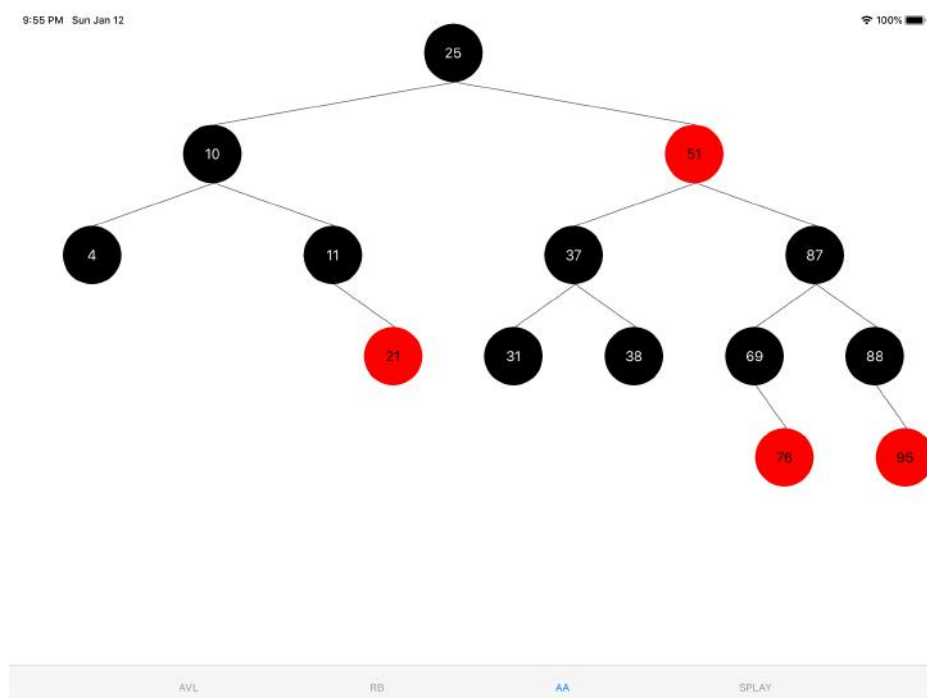
Após concluir as alterações no nó, é verificado se o nó tem filhos à esquerda e à direita. Caso alguma dessas condições seja confirmada, é desenhada uma aresta ligando o nó até onde será desenhado o filho e é feita a chamada recursiva passando como parâmetros o filho encontrado e a área onde este será desenhado.

Figura 23 – Detalhes de um nó após um toque prolongado



Na função de desenhar nós de uma árvore AA, como este não guarda sua cor, é passado um parâmetro extra para a definição de sua cor. Inicialmente é usada a cor preta para a raiz e depois é determinada a cor dos filhos de cada nó, de acordo com a diferença de nível entre eles e seu pai. Para um filho à esquerda, sempre é determinado que sua cor é preta, pois não existem filhos à esquerda vermelhos. Para um filho à direita, o nó é vermelho caso seu nível seja igual ao nível do nó atual, assumindo a cor preta caso contrário.

Figura 24 – Exemplo de árvore AA



Para a realização de uma busca é chamada outra função recursiva, similar à função de desenhar a árvore, onde é passado mais um parâmetro, o elemento buscado, que é um número inteiro que representa o identificador de um nó.

Caso o nó atual tenha o identificador igual ao elemento buscado, o nó recebe uma borda verde, representando que o elemento foi encontrado. Caso contrário, ele continua desenhando a árvore e fazendo uma verificação se o elemento buscado é superior ou inferior ao identificador do nó atual.

Caso o elemento buscado seja inferior ao identificador atual e o nó tenha um filho à esquerda, é indicado que a busca continua e a ligação entre os nós é realçada usando a cor verde, como ilustrado na figura 21. A função então chama a si própria para o filho esquerdo e é chamada a função de desenho fora da busca para o filho direito, se este existir.

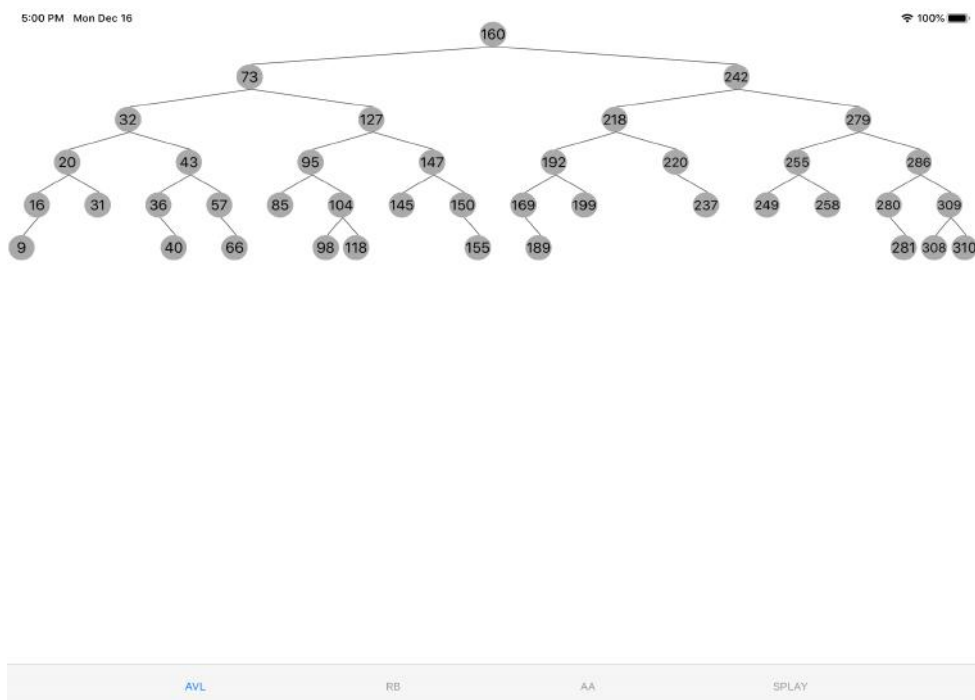
Caso o elemento buscado seja maior que o identificador atual, a função chama a si própria para o filho direito e é chamada a função de desenho fora da busca para o filho esquerdo, se este existir.

Caso um nó não tenha o filho que dá prosseguimento à função de busca, esta é encerrada e é exibido um alerta avisando ao usuário que o nó procurado não foi encontrado e não faz parte da árvore, como mostrado na figura 22. A função de desenho fora da busca continua sendo chamada para seu outro filho, se este existir.

Quando uma árvore atinge a altura 5, os nós da árvore desenhada são reduzidos para se ter uma melhor visualização da estrutura da árvore. Essa redução acontece novamente

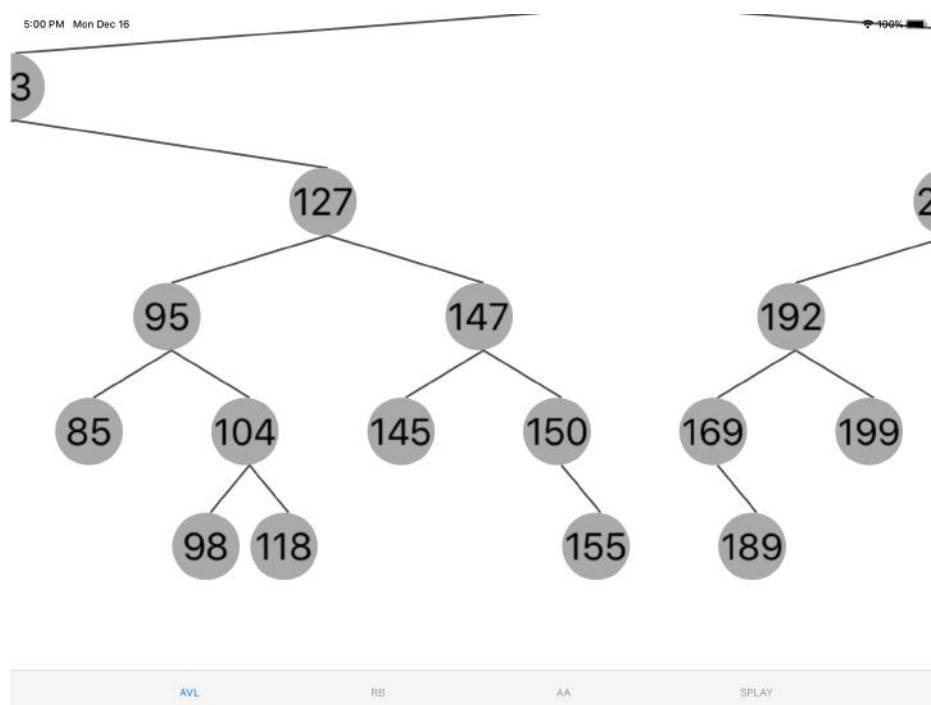
quando a árvore atinge a altura 6, onde os nós atingem o tamanho mínimo, sendo ainda possível a visualização da árvore, porém a leitura dos identificadores seria dificultada caso os nós fossem reduzidos ainda mais. Quando a altura da árvore é reduzida, os nós aumentam de tamanho na mesma proporção.

Figura 25 – Exemplo de uma árvore AVL com altura 6



Para facilitar a leitura e interação, foi implementada uma função no scrollview para que seus elementos possam ser ou ampliados e visualizados facilmente ou reduzidos para melhor observação da estrutura da árvore. Para ampliar o conteúdo da tela é necessário apenas fazer o gesto de afastar dois dedos tocando a tela e para reduzir o conteúdo é só aproximar os dedos.

Figura 26 – Exemplo de uma árvore AVL com ampliação dos nós



5 CONCLUSÃO

Neste trabalho foram apresentadas versões interativas da representação visual de diferentes árvores de busca balanceadas e suas operações de balanceamento. A aplicação mostra que a visualização da árvore utilizando uma interface intuitiva pode melhorar o entendimento de como a lógica das operações de balanceamento funcionam em uma árvore balanceada.

A árvore que teve a implementação mais fácil foi a árvore AVL, pois não houve nenhuma particularidade que dificultasse a escrita do código. As árvores Rubro-Negra e Splay têm a particularidade de o ponteiro do nó apontar para si mesmo caso não exista um filho à esquerda ou à direita, sendo necessária uma checagem extra, dificultando o código.

A árvore AA, apesar de ter uma lógica simples, teve o algoritmo de desenho mais complexo por não guardar sua cor, mas seu nível. Desta forma, foi necessário checar o nível do nó pai e comparar com o nível do nó filho em todos os passos à direita no algoritmo, já que não há filhos vermelhos à esquerda nesse tipo de árvore.

Com exceção da árvore Splay, as demais árvores apresentam uma boa visualização em uma tela de iPad. A árvore splay, por ter sua altura normalmente maior que as demais, mesmo com uma quantidade reduzida de nós, apresenta dificuldade de visualização de seus nós, principalmente se os nós forem buscados ou inseridos sequencialmente.

Todo o código para a implementação do aplicativo pode ser encontrado no repositório do GitHub no endereço <<https://github.com/marcosamorimrc/BalancedBinaryTrees>>.

5.1 TRABALHOS FUTUROS

A inserção manual de nós é satisfatória para a montagem de árvores e para acompanhamento de suas rotações e uma montagem aleatória é boa para testes com árvores grandes, mas poderia haver alguma forma, em trabalhos futuros, de incluir nós pré-determinados de algum arquivo de fora do aplicativo em diferentes árvores para a comparação de resultados finais.

Para melhorar a comparação dos resultados, seria interessante ao usuário que houvesse a opção de armazenar na memória do aparelho uma cópia da árvore atual, possibilitando a montagem de variações da árvore armazenada. Também poderia ter como guardar todas as operações que foram executadas para a montagem de uma árvore para a comparação entre diferentes tipos de árvore.

Por último, para facilitar a visualização das rotações, poderia ser executada animações dos nós rotacionando até a árvore atingir o estado de balanceada.

REFERÊNCIAS

ANDERSSON, A. Balanced search trees made simple. **Workshop on Algorithms and Data Structures**, p. 60–71, 1993.

BALANCEDBINARYTREES. 2019. Disponível em: <<https://github.com/marcosamorimrc/BalancedBinaryTrees/>>.

CORBET. **Schedulers: the plot thickens**. 2007. Disponível em: <<https://lwn.net/Articles/230574/>>.

CORMEN, T. H. et al. **Algoritmos. Teoria e Prática Tradução da Segunda Edição Americana**. [S.l.]: Campus, 2002.

CPLUSPLUS. **Map em C++**. 2019. Disponível em: <<http://www.cplusplus.com/reference/map/map/>>.

DASGUPTA, S.; PAPADIMITRIOU, C.; VAZIRANI, U. **Algoritmos**. [S.l.]: AMGH Editora, 2009.

GOODRICH, M. T.; TAMASSIA, R. **Algorithm Design: Foundations, Analysis and Internet Examples**. [S.l.]: John Wiley and Sons, 2001.

ROMA&WEISS'SCODE. 2019. Disponível em: <<http://orion.lcg.ufrj.br/roma/orgdados/index.html>>.

ROSEN, K. H. **Matemática Discreta e suas Aplicações**. [S.l.]: Grupo A Educação, 2009.

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. [S.l.]: Addison-Wesley Professional, 2011.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de Dados e seus Algoritmos**. [S.l.]: Ltc, 1994.

WIKIPEDIA. **Binary search tree**. 2019. Disponível em: <https://en.wikipedia.org/wiki/Binary_search_tree>.

WIKIPEDIA. **Scheduling (computing)**. 2019. Disponível em: <[https://en.wikipedia.org/wiki/Scheduling_\(computing\)](https://en.wikipedia.org/wiki/Scheduling_(computing))>.